ΕΠΛ 602:Foundations of Internet Technologies

Cloud Computing

Outline

Bigtable (data component of cloud)

Web search

based on ch13 of the web data book

What is Cloud Computing?

A Cloud is

- an infrastructure, transparent to the end-user,
- which is used by a company or organization to provide services to its customers
- via ***network***
- where the infrastructure resources are used ***elastically*** and the
- customer is ***charged according to usage***.

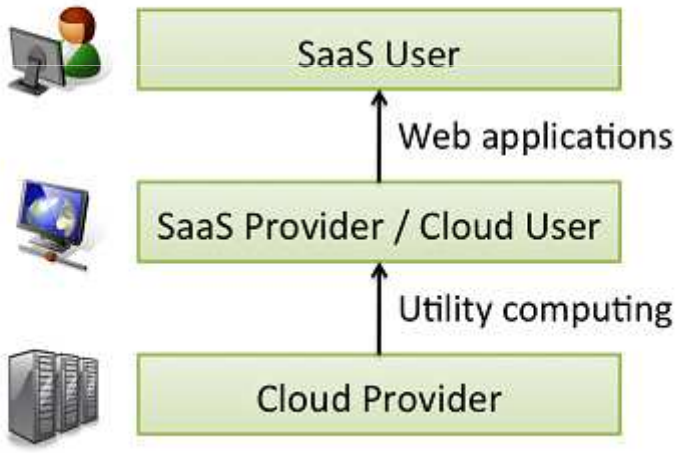
under the hood: large clusters of commodity hardware
terms: virtualization, elasticity, utility computing, pay-as-you-go

What is Cloud Computing?

Cloud Computing:

- the applications delivered as services over the Internet (SaaS) and
- the hardware and systems software in the datacenters that provide these services (a Cloud)

roles of the people as *users* or *providers* of these layers of Cloud Computing



What is Cloud Computing?

What is offered:

- Data Storage

  Examples:

  AWS Simple Storage System (S3)

- Infrastructure as a Sevice (IaaS)

  Provide computing instances (e.g., servers running Linux) as a service.

  Examples:

  AWS Elastic Computing Cloud (EC2).

- Platform as a Service (PaaS)

  The delivery of a computing platform and solution stack as a service via network.

  Examples:

  Google AppEngine

- Software as a Service (SaaS)

  Software that is deployed over network as a service.

  Examples:

  Google Documents

  Google Calendar

  Google Reader

What is Cloud Computing?

Some Cloud Providers

- Amazon Web Services (AWS)
  http://aws.amazon.com/
  The most complete set of Cloud services.
- Google App Engine
  http://code.google.com/appengine/
- IBM Cloud
  http://www.ibm.com/ibm/cloud/
- Microsoft Windows Azure
  http://www.microsoft.com/windowsazure/

Map Reduce

A programming paradigm that comes with a framework to provide to the programmers an easy way for parallel and distributed computing.

- Designed by Google (published in 2004)
- Designed to scale well on large clusters --> Perfect for Cloud Computing.
- Input & output data stored in a distributed file system.
- Fault tolerance, status & monitoring tools
- It is attractive because it provides a simple model.
- More than 10,000 distinct MapReduce programs have been implemented in Google.
    - Graph processing, text processing, machine learning, statistical machine translation etc
- Open source implementation (Hadoop)

MapReduce

- Programmers specify two functions:

  map (k, v) → <k', v'>*

  Takes an input pair and produces an intermediate (key, value) pair

  reduce (k', v') → <k', v'>*

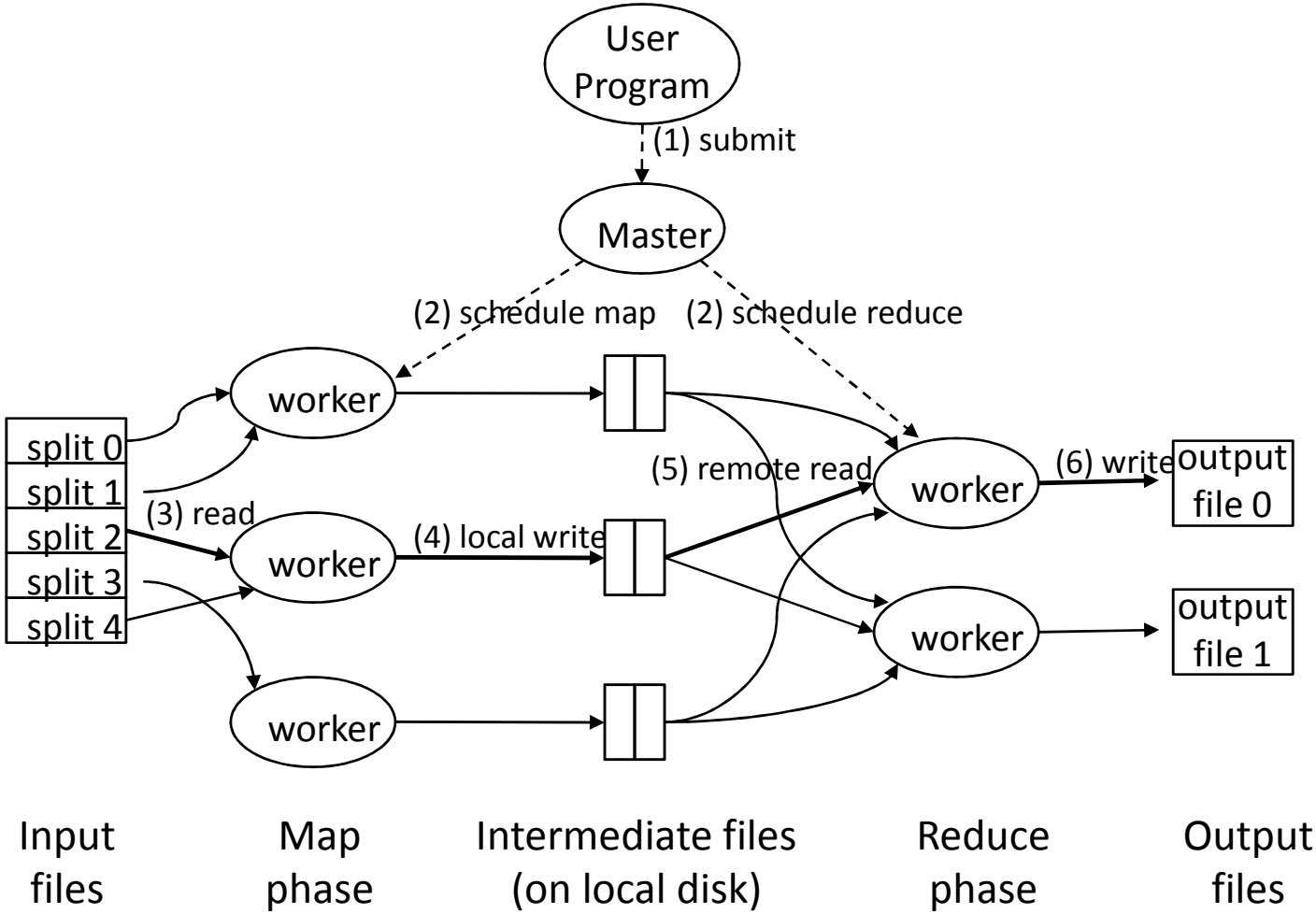
  - *All values with the same key are sent to the same reducer*

  Each reducer accepts as input all values associated with the same intermediate key and merges them together to produce a possibly smaller set

  The intermediate values are supplied to reduce function via an iterator

- The execution framework handles everything else…

MapReduce Overall Architecture



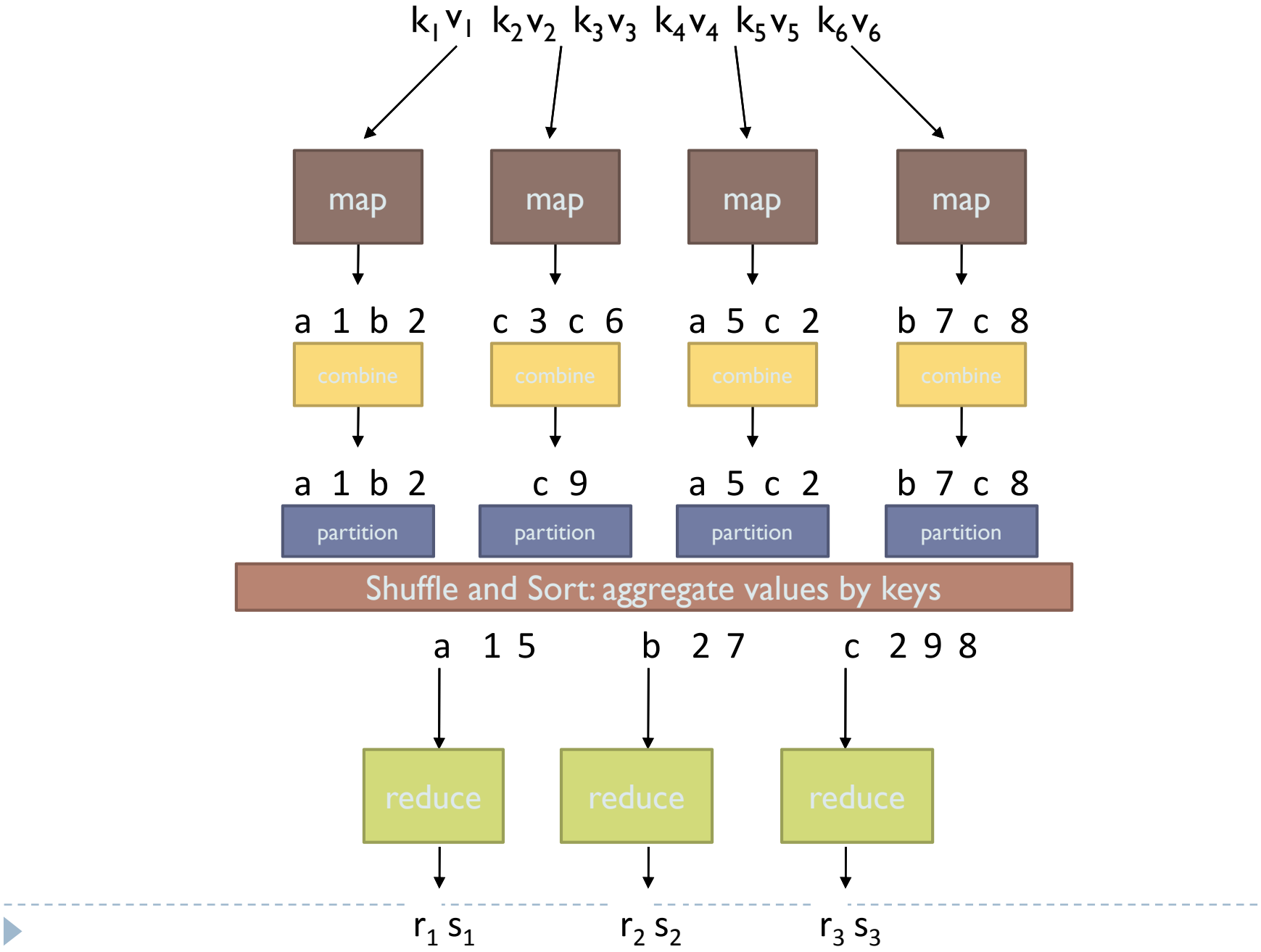
Adapted from (Dean and Ghemawat, OSDI 2004)

MapReduce "Runtime"

- Handles scheduling
  - Assigns workers to map and reduce tasks
- Handles "data distribution"
  - Moves processes to data
- Handles synchronization
  - Gathers, sorts, and shuffles intermediate data
- Handles errors and faults
  - Detects worker failures and automatically restarts
- Handles speculative execution
  - Detects "slow" workers and re-executes work
- Everything happens on top of a distributed FS

MapReduce

- Programmers specify two functions:
  map (k, v) → <k', v'>*
  reduce (k', v') → <k', v'>*
  - All values with the same key are reduced together
- The execution framework handles everything else…
- Not quite…usually, programmers also specify:
  partition (k', number of partitions) → partition for k'
  - Often a simple hash of the key, e.g., hash(k') mod R
  - Divides up key space for parallel reduce operations
  combine (k', v') → <k', v'>*
  - Mini-reducers that run in memory after the map phase
  - Used as an optimization to reduce network traffic


k1 v1 k2 v2 k3 v3 k4 v4 k5 v5 k6 v6
map
map
map
map
a 1 b 2
c 3 c 6
a 5 c 2
b 7 c 8
combine
combine
combine
combine
a 1 b 2
c 9
a 5 c 2
b 7 c 8
partition
partition
partition
partition
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 9 8
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

MapReduce can refer to…

- The programming model
- The execution framework (aka “runtime”)
- The specific implementation

MapReduce Implementations

- Google has a proprietary implementation in C++
  - Bindings in Java, Python
- Hadoop is an open-source implementation in Java
  - Development led by Yahoo, used in production
  - Now an Apache project
  - Rapidly expanding software ecosystem, but still lots of room for improvement
- Lots of custom research implementations
  - For GPUs, cell processors, etc.

Distributed File System

- Don't move data to workers… move workers to the data!
  - Store data on the local disks of nodes in the cluster
  - Start up the workers on the node that has the data local
- Why?
  - Network bisection bandwidth is limited
  - Not enough RAM to hold all the data in memory
  - Disk access is slow, but disk throughput is reasonable
- A distributed file system is the answer
  - GFS (Google File System) for Google's MapReduce
  - HDFS (Hadoop Distributed File System) for Hadoop

GFS: Assumptions

- Choose commodity hardware over "exotic" hardware
  - Scale "out", not "up"
- High component failure rates
  - Inexpensive commodity components fail all the time
- "Modest" number of huge files
  - Multi-gigabyte files are common, if not encouraged
- Files are write-once, mostly appended to
  - Perhaps concurrently
- Large streaming reads over random access
  - High sustained throughput over low latency

GFS: Design Decisions

- Files stored as chunks
  - Fixed size (64MB)
- Reliability through replication
  - Each chunk replicated across 3+ chunkservers
- Single master to coordinate access, keep metadata
  - Simple centralized management
- No data caching
  - Little benefit due to large datasets, streaming reads
- Simplify the API
  - Push some of the issues onto the client (e.g., data layout)

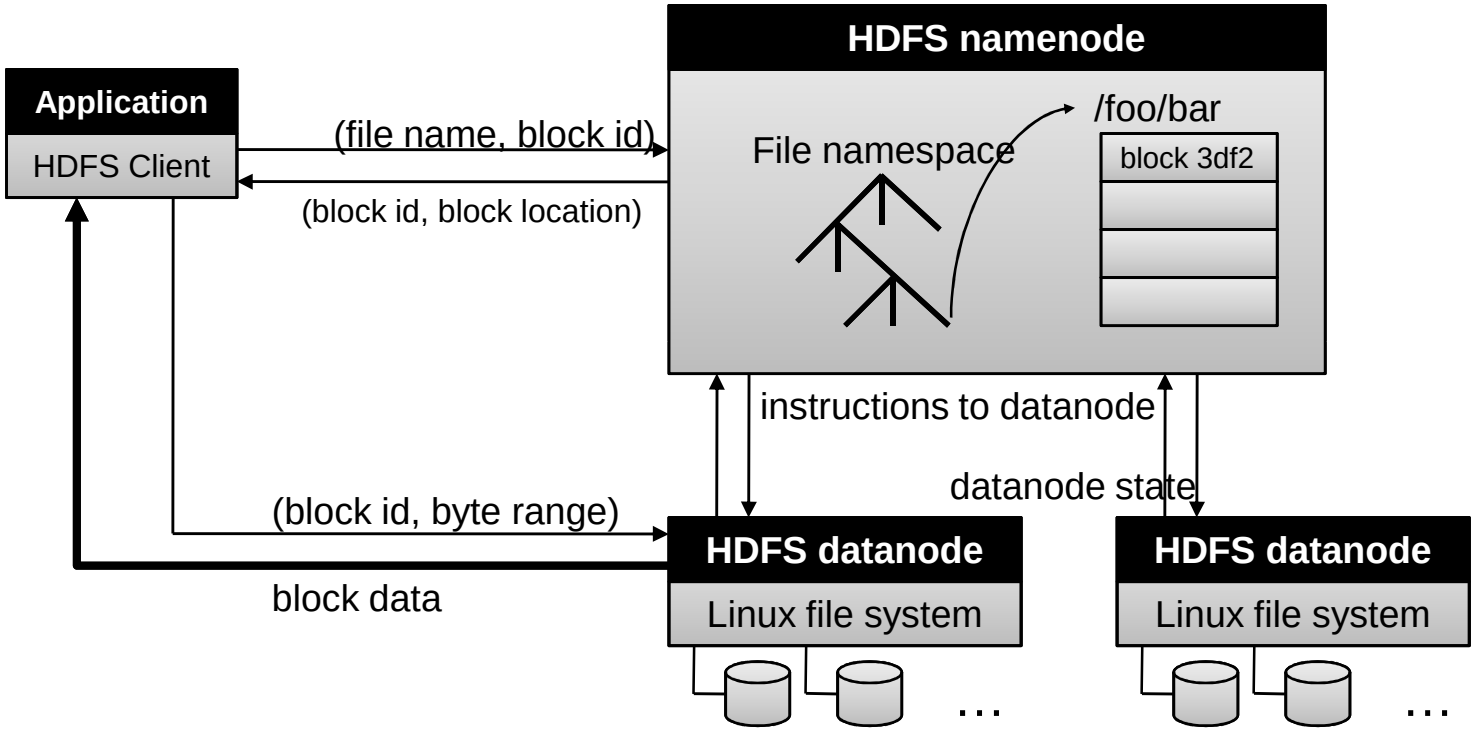
From GFS to HDFS

- Terminology differences:
  - GFS master = Hadoop namenode
  - GFS chunkservers = Hadoop datanodes
- Functional differences:
  - No file appends in HDFS (planned feature)
  - HDFS performance is (likely) slower

Namenode Responsibilities

- Managing the file system namespace:
  - Holds file/directory structure, metadata, file-to-block mapping, access permissions, etc.
- Coordinating file operations:
  - Directs clients to datanodes for reads and writes
  - No data is moved through the namenode
- Maintaining overall health:
  - Periodic communication with the datanodes
  - Block re-replication and rebalancing
  - Garbage collection

HDFS Architecture



MapReduce/GFS Summary

- Simple, but powerful programming model
- Scales to handle petabyte+ workloads
  - Google: six hours and two minutes to sort 1PB (10 trillion 100-byte records) on 4,000 computers
  - Yahoo!: 16.25 hours to sort 1PB on 3,800 computers
- Incremental performance improvement with more nodes
- Seamlessly handles failures, but possibly with performance penalties

Bigtable

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber: ***Bigtable: A Distributed Storage System for Structured Data,*** *OSDI 2006*

Data Model

- A table in Bigtable is a sparse, distributed, persistent multidimensional sorted map
- Map indexed by a row key, column key, and a timestamp
  - (row:string, column:string, time:int64) → uninterpreted byte array
- Supports lookups, inserts, deletes
  - Single row transactions only

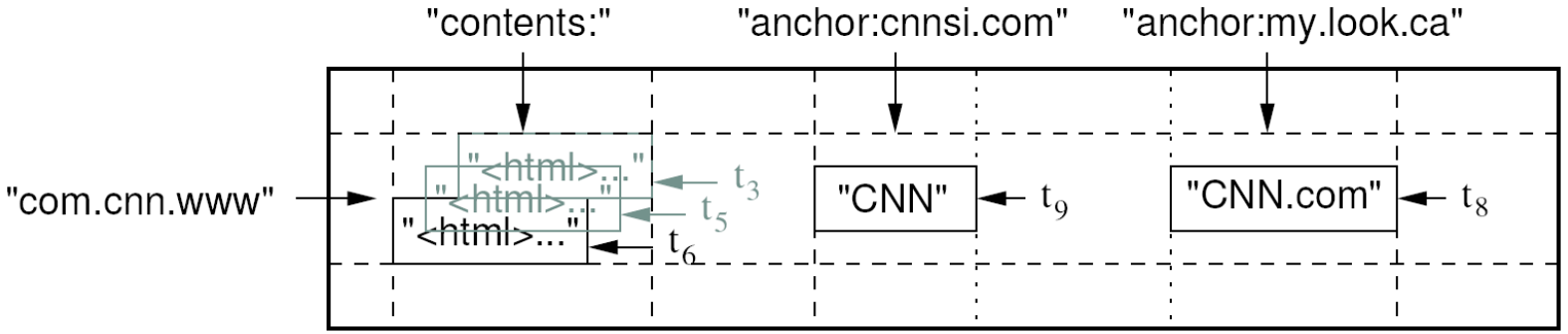


Image Source: Chang et al., OSDI 2006

Rows and Columns

- Rows maintained in sorted lexicographic order
  - Applications can exploit this property for efficient row scans
  - Row ranges dynamically partitioned into tablets (serve as units of distribution and load balance)

Reads of short row ranges typically communication with a small number of machines.

Exploit, this for example, pages in the same domain grouped together into contiguous rows by reversing the hostname components of the URLs.
For example, store data for maps.google.com/index.html under key com.google.maps/index.html. makes some host and domain analyses more efficient

Read/write on a single row is atomic

Rows and Columns

- Columns grouped into column families
  - Stored same type of data
  - Created before any key stored – after created any key can be used – Unbounded number of columns

Column key = family:qualifier

- example families: language, anchor
- anchor:my.look.ca -> cell contains the text of the link

Access control and disk and memory accounting

- Column families provide locality hints – multiple column families can be grouped together to form a locality group

Timestamps

- *Each cell can contain multiple versions of the same data* indexed by timestamp.
- Timestamps can be assigned by Bigtable (real time in microseconds,) or by client applications.
- Different versions of a cell are stored in decreasing timestamp order, so that the most recent versions can be read first.

- Two per-column-family settings for garbage-collection:

  only the last n versions of a cell, or

  only new-enough versions (e.g., only values written in the last 7 days).

Example: the timestamps of the crawled pages in the contents: column equal to the times at which these page versions were crawled.

APIs

- for creating and deleting tables and column families
- for changing cluster, table, and column family metadata, such as access control rights
- to look up values from individual rows, or iterate over a subset of the data in a table.
- iterate over multiple column families -- several mechanisms for limiting the rows, columns, and timestamps produced by a scan

```
// Open the table
Table *T = OpenOrDie("/bigtable/web/webtable");

// Write a new anchor and delete an old anchor
RowMutation r1(T, "com.cnn.www");
r1.Set("anchor:www.c-span.org", "CNN");
r1.Delete("anchor:www.abc.com");
Operation op;
Apply(&op, &r1);
```

Figure 2: Writing to Bigtable.

```
Scanner scanner(T);
ScanStream *stream;
stream = scanner.FetchColumnFamily("anchor");
stream->SetReturnAllVersions();
scanner.Lookup("com.cnn.www");
for (; !stream->Done(); stream->Next()) {
  printf("%s %s %lld %s\n",
         scanner.RowName(),
         stream->ColumnName(),
         stream->MicroTimestamp(),
         stream->Value());
}
```

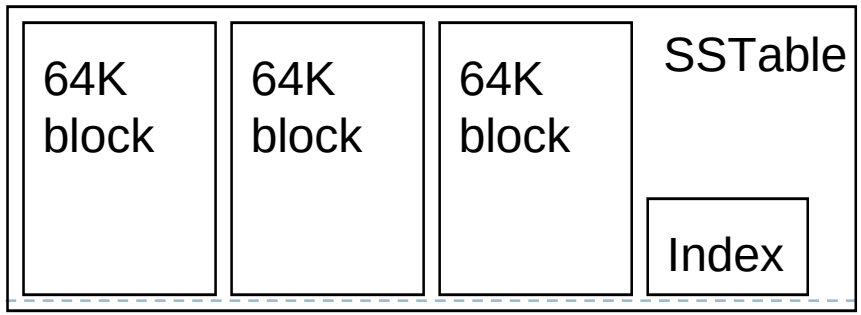
Figure 3: Reading from Bigtable.

Bigtable Building Blocks

- GFS
- Chubby (highly available distributed lock service)
- SSTable (file format used internally to store Bigtable data)

SSTable

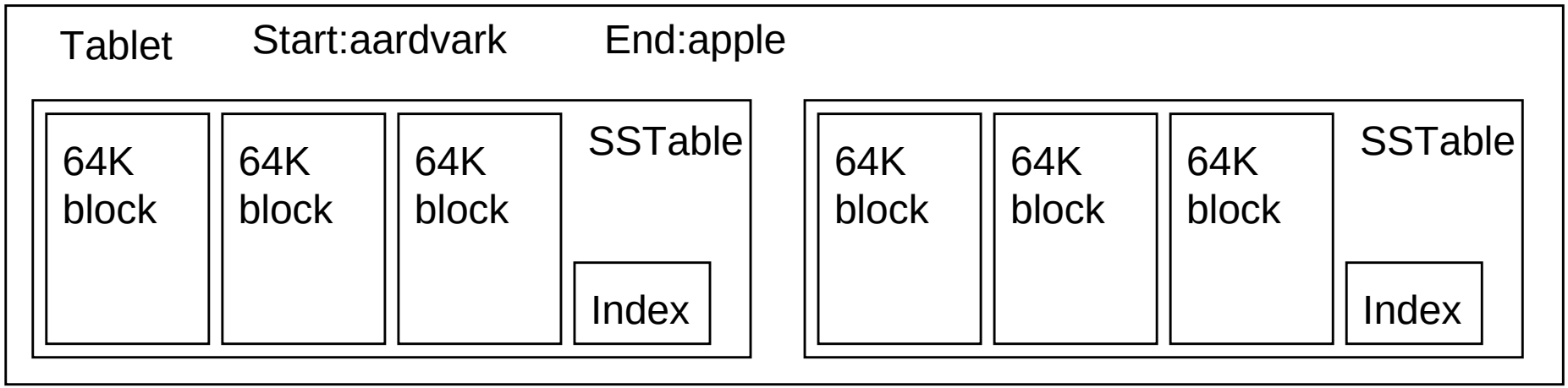
- Basic building block of Bigtable
- Persistent, ordered immutable map from keys to values
  - Stored in GFS
- Sequence of blocks on disk plus an index for block lookup
  - Can be completely mapped into memory
- Supported operations:
  - Look up value associated with key
  - Iterate key/value pairs within a key range



Source: Graphic from slides by Erik Paulson

Tablet

- Initially one tablet per table, as the table grows -> automatically split into tablets (dynamically partitioned range of rows)
- Built from multiple SSTables



Source: Graphic from slides by Erik Paulson

Architecture

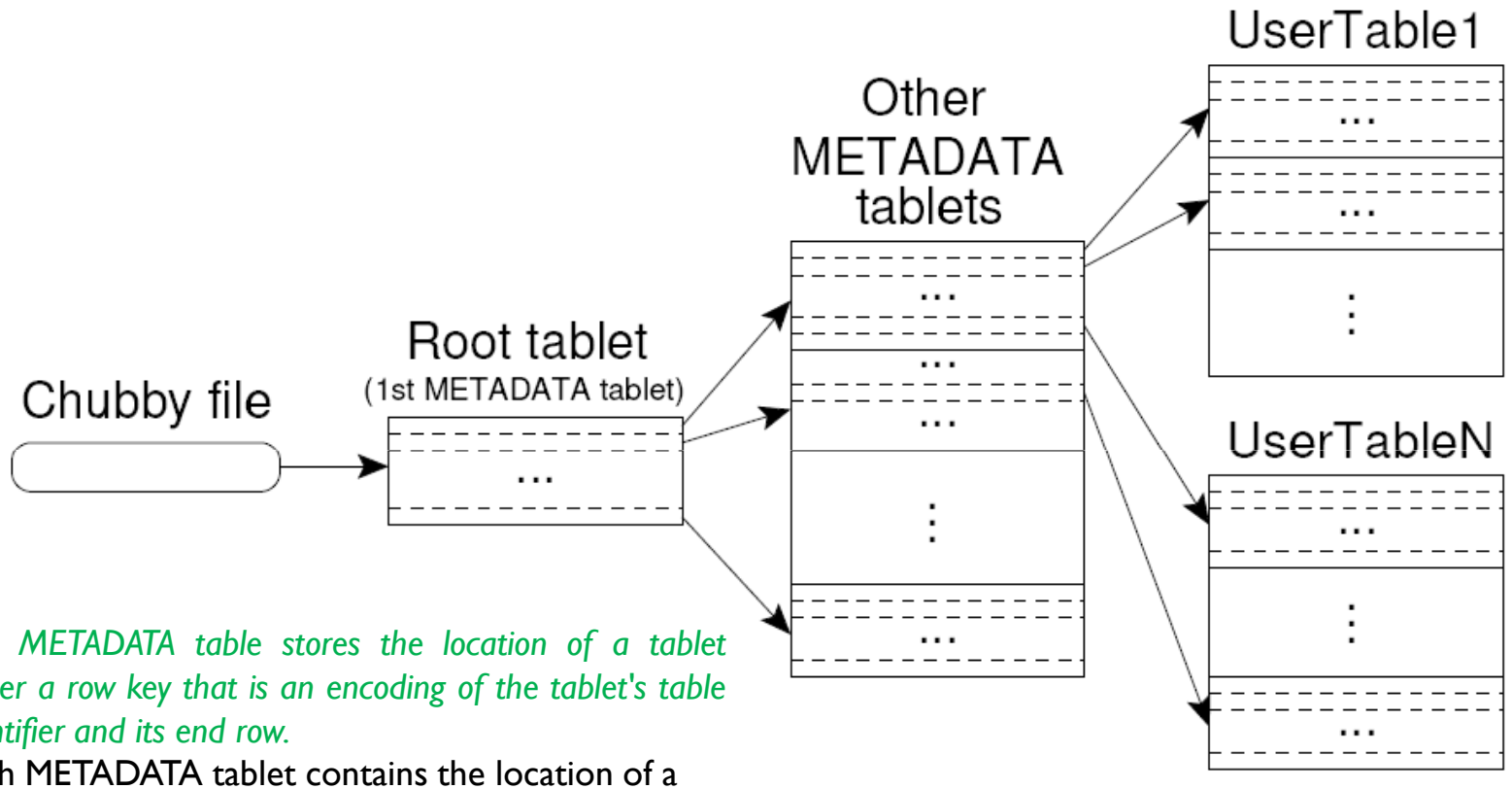
- Client library
- Single master server
- Tablet servers

Bigtable Tablet Servers

- Each tablet server manages a set of tablets
  - Typically between ten to a thousand tablets
  - Each 100-200 MB by default
- Handles read and write requests to the tablets
- Splits tablets that have grown too large

Clients communicate directly with tablets

Tablet Location



The METADATA table stores the location of a tablet under a row key that is an encoding of the tablet's table identifier and its end row.

Each METADATA tablet contains the location of a set of user tablets

Root tablet contains the location of all tablets in the metadata table (never split)

1st level contains the location of the Root tablet

The client library caches tablet locations.

Image Source: Chang et al., OSDI 2006

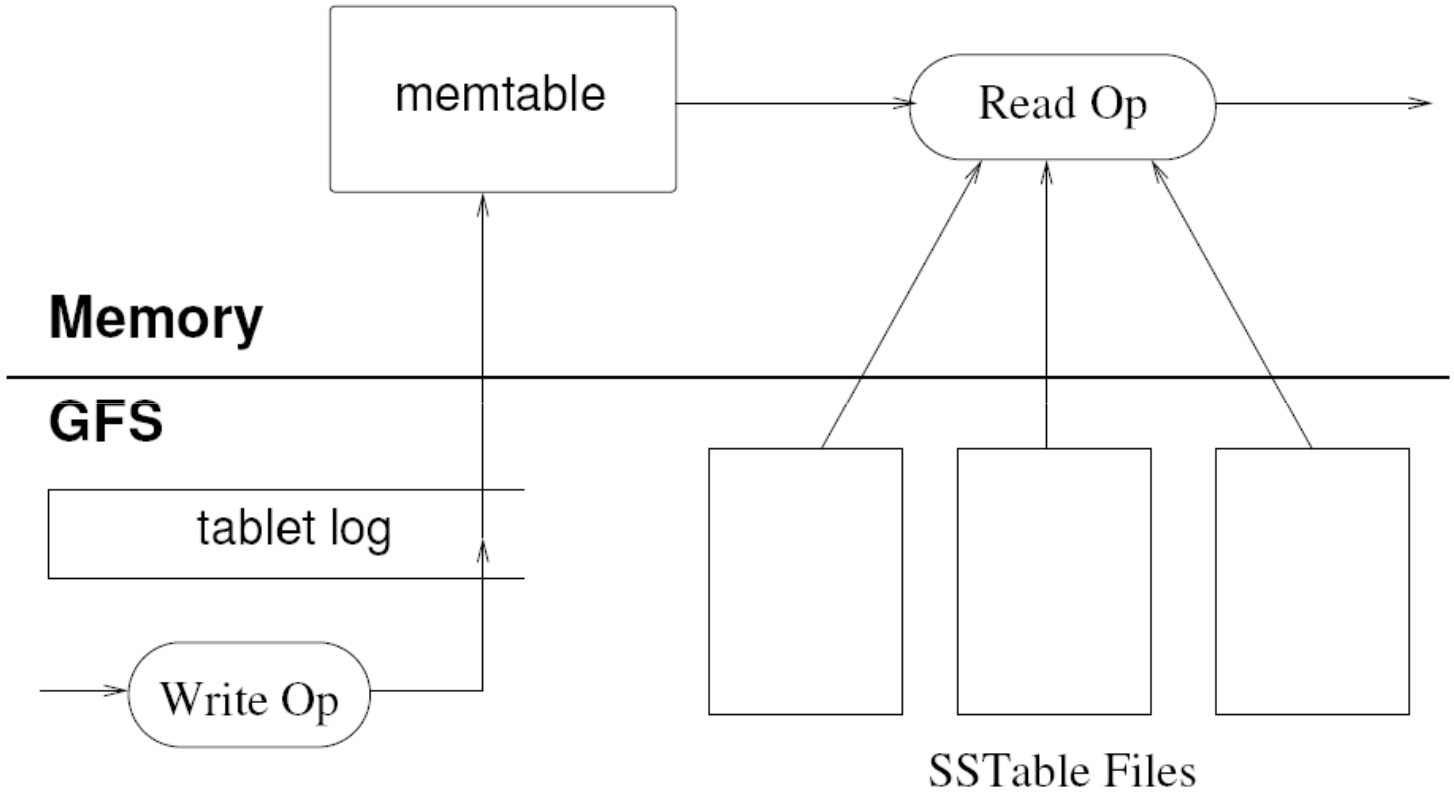
Tablet Assignment

- Master keeps track of:
  - Set of live tablet servers
  - Assignment of tablets to tablet servers
  - Unassigned tablets
- Each tablet is assigned to one tablet server at a time
  - Tablet server maintains an exclusive lock on a file in Chubby
  - Master monitors tablet servers and handles assignment
- Changes to tablet structure
  - Table creation/deletion (master initiated)
  - Tablet merging (master initiated)
  - Tablet splitting (tablet server initiated)

Bigtable Master

- Balances tablet server load. Tablets are distributed randomly on nodes of the cluster for load balancing.
- Handles garbage collection
- Handles schema changes

Tablet Serving



Persistent state is stored in GFS
Updates in a commit log that stores redo records
Recently committed ones in memtable

Image Source: Chang et al., OSDI 2006

Compactions

- Minor compaction
  - Converts the memtable into an SSTable
  - Reduces memory usage and log traffic on restart
- Merging compaction
  - Reads the contents of a few SSTables and the memtable, and writes out a new SSTable
  - Reduces number of SSTables
- Major compaction
  - Merging compaction that results in only one SSTable
  - No deletion records, only live data

Lock server

- Chubby
  - Highly-available & persistent distributed lock service
  - Five active replicas; one acts as master to serve requests
- Chubby is used to:
  - Ensure there is only one active master
  - Store bootstrap location of BigTable data
  - Discover tablet servers
  - Store BigTable schema information
  - Store access control lists
- If Chubby dies for a long period of time… Bigtable dies too….
  - But this almost never happens…

Optimizations

- Log of tablets in the same server are merged in one log per tablet server (node)
- *Locality groups: separate SSTables are created for each locality group of column families that form the locality groups.*
- Use efficient and lightweight compression to reduce the size of SSTable blocks. Since data are organized by column(s) very good compression is achieved (similar values together)
- Tablet servers use two levels of caching (Scan Cache caches key-value pairs returned by the SSTable interface to the tablet server code – Block Cache caches SSTbables blocks read by the GFS)
- Bloom filters are used to skip some SSTables and reduce read overhead.

HBase

- Open-source clone of Bigtable
- Implementation hampered by lack of file append in HDFS

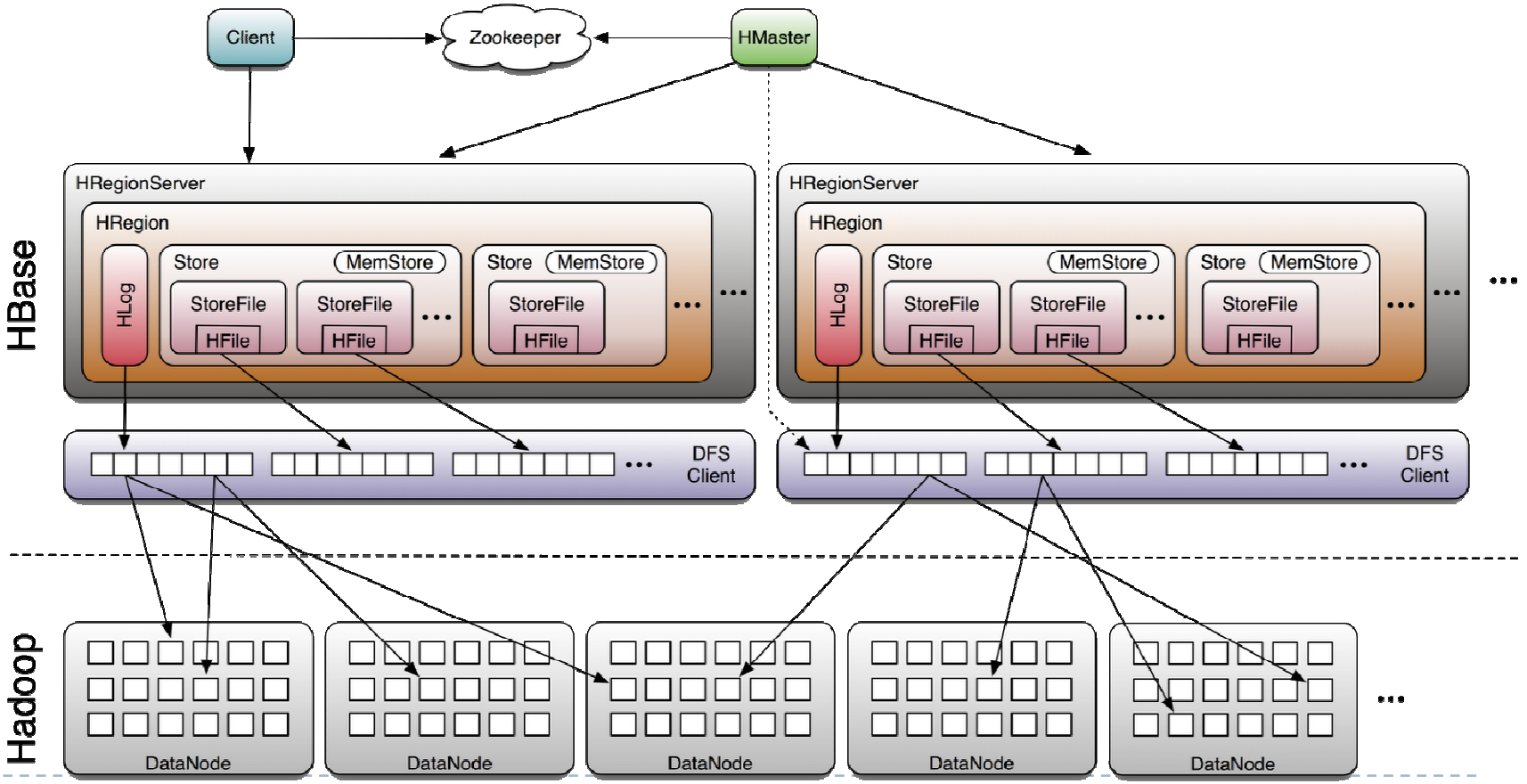


Image Source: http://www.larsgeorge.com/2009/10/hbase-architecture-101-storage.html

The Facebook MapReduce Data Warehouse System[8]

- Facebook was one of the first enterprises that implemented a large data warehouse system using MapReduce technology rather than a DBMS.
- 2.5PB of data managed by Hadoop (04/2009).
- Facebook's Hadoop/Hive system ingests 15 TB of new data per day.
- Facebook has the second largest Hadoop installation: 610 nodes in one cluster (now maybe more than 1,000 nodes).

[8] http://www.dbms2.com/2009/04/15/cloudera-presents-the-mapreduce-bull-case/

Image Source: http://www.larsgeorge.com/2009/10/hbase-architecture-101-storage.html

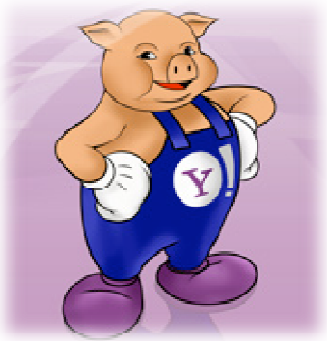
Need for High-Level Languages

- Hadoop is great for large-data processing!
  - But writing Java programs for everything is verbose and slow
  - Not everyone wants to (or can) write Java code
- Solution: develop higher-level data processing languages
  - Hive: HQL is like SQL
  - Pig: Pig Latin is a bit like Perl (relation-algebra like

Jaql from IBM
Pig reported to be used for over 60% of Yahoo!'s MapReduce use cases
Hive reported to be used for over 90%of the Facebook use cases

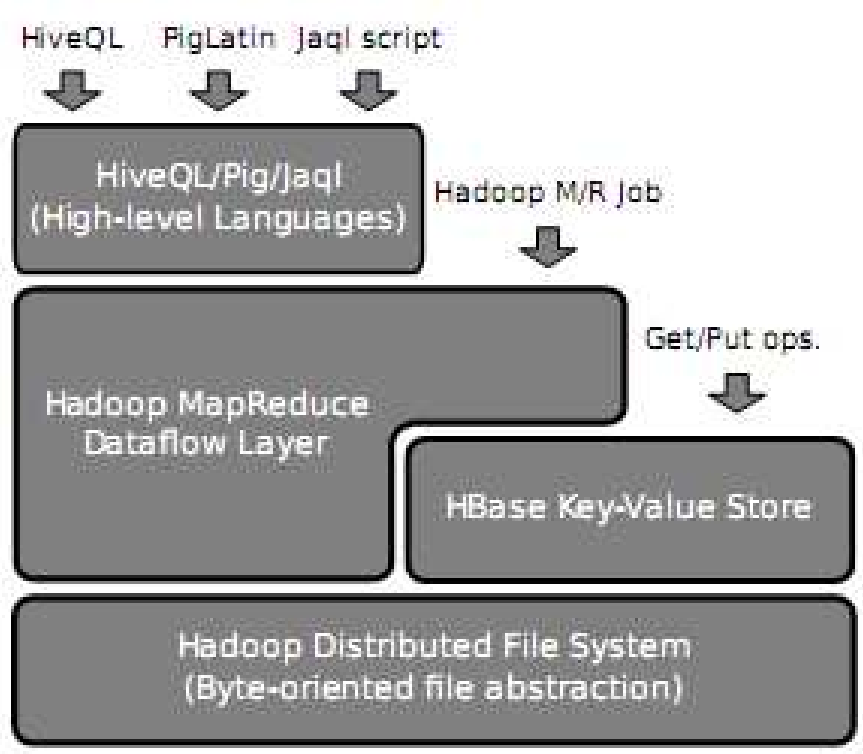
Hive and Pig

- Hive: data warehousing application in Hadoop
  - Query language is HQL, variant of SQL
  - Tables stored on HDFS as flat files
  - Developed by Facebook, now open source
- Pig: large-scale data processing system
  - Scripts are written in Pig Latin, a dataflow language
  - Developed by Yahoo!, now open source
  - Roughly 1/3 of all Yahoo! internal jobs



- Common idea:
  - Provide higher-level language to facilitate large-data processing
  - Higher-level language "compiles down" to Hadoop jobs

Hive and Pig



Layers in the software architecture of Hadoop stack.

For batch analytics, the middle layer is the Hadoop MapReduce system, *which applies map operations to the data in partitions of an HDFS file, sorts and redistributes the results based on key values in the output data, and then performs reduce operations on the groups of output data items with matching keys from the map phase of the job.*

For applications just needing basic key-based record management operations, the HBase store as a key-value layer (*The contents of HBase can either be directly accessed and manipulated by a client application or accessed via Hadoop for analytical needs*.

Clients of the Hadoop stack may use of a declarative language over the bare MapReduce programming model.

Questions?